

Web 2.0 und Sicherheit



<https://iuk.one/1066-1011>

Clemens H. Cap

ORCID: 0000-0003-3958-6136

Department of Computer Science
University of **Rostock**
Rostock, Germany
clemens.cap@uni-rostock.de

Version 1



1. Übersicht

Um was geht es bei Web 2.0 Sicherheit?

1. Übersicht

2. Mobile Code

3. Injection

4. Cross Site

5. Trusted IO Problem

1. Übersicht

Übersicht

Web 2.0-unabhängige Probleme

- In OS, Kommunikations-Stack, Bibliotheken.
- Auf Server & Infrastruktur.
- Fehlverhalten des Benutzers.

Technologie-Probleme (JS, CSS, DOM, HTML, Server)

- **Mobile Code:** Ausführung dynamisch, vom Server geladenen Codes.
- **Cross Site:** Von welcher Site kommt welcher Code.
- **Trusted IO:** Vertrauen in Ein- und Ausgabe.
- **Client Access:** Zugriff auf Camera, Mike, Storage
- **Meta Themen:** Chrome vs Web, Zeichen vs Meta-Zeichen (Injection)

Konzept-Probleme:

- **Identität:** Multiple, fake, Diebstahl.
- **Reputation:** Vergrößert / vermindert; Sybil, Profile Poisoning.
- **Inhalt:** Content Protection, Screen Scraping, Remixing, Reuse.

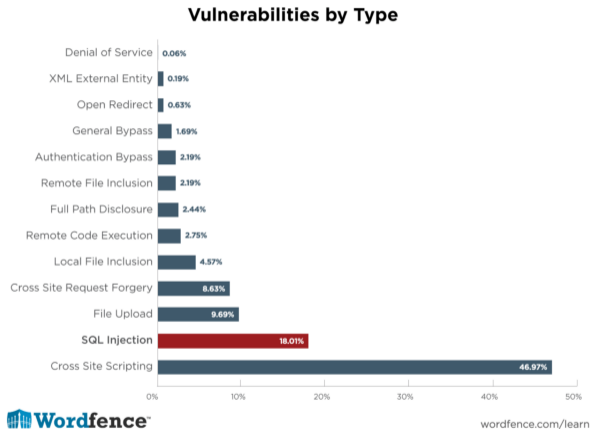


Abb. 1: © Rechte siehe Anhang.

1. Übersicht

Klassifikation von Sicherheitsproblemen

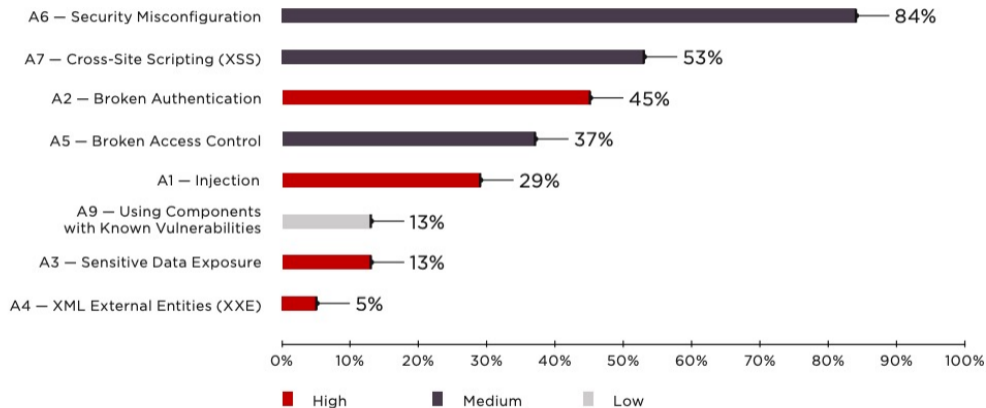


Abb. 2: © Rechte siehe Anhang.

Ziele des Angreifers

Abstrakte Kategorien:

- Finanzielle Ziele
- Verhaltens-Ziele
- Überwachungsziele
- Schädigungs-Ziele

Konkrete Ziele:

- Ausspähen von Passwörtern, Daten und Zugangswegen
- Modifikation von Transaktionsdaten
- Installation von Mal- und Spyware
- Gewinnen von Benutzer-Profilen.
- Kontrolle von Benutzer-Verhalten
- Übernehmen von Rechnern
- Website Defacing
- Profile Poisoning
- Link Spamming

2. Mobile Code

Die Problematik des mobilen Codes
und Lösungsansätze dazu.

1. Übersicht
2. Mobile Code
3. Injection
4. Cross Site
5. Trusted IO Problem

Sicherheitsproblematik bei Mobilem Code

Problematik

Von Dritten eingebrachter Code führt mit zu weitgehenden Rechten unerwünschte Funktionen aus.

Beispiele:

- Installationen und Updates.
- Datenträger Autorun (USB, DVD).
- Macros in Dokumenten.
- Mobiler Code.

Technologien mobilen Codes: Java, ActiveX, Javascript, VBScript

Lösungsansätze:

- 1 Sandbox
- 2 Signierter Code
- 3 Policy Files

Sandbox

Mobiler Code läuft innerhalb einer Sandbox.
Die Sandbox verhindert (bestimmte) Zugriffe auf lokale Ressourcen.

Beispiel: Kein Plattenzugriff.

- **Problem:** Kein clientseitiger Status speicherbar.
- **Also:** Cookies.
- **Folge:** Profiling & Tracking

Beispiel: Kein Netzzugriff auf außerhalb der Domäne, von der Code bezogen.

- **Grund:** Exfiltrieren von Information verbieten.
- **Problem:** Kein sinnvoller Zugriff auf Dritt-Ressourcen.

2. Mobile Code Sandbox (2)

Idee 1: Kontrolliertes Durchbrechen bestehender Sandbox-Mechanismen.

Idee 2: APIs einführen, die nur kontrollierten Zugang erlauben.

Beispiele für APIs:

- Spezifische second / third party Cookie Limitierungen.
- Same Origin Sandbox für IndexedDB, localStorage.
- Spezifische *non-extractable private key stores*.
- Server Header kann Cross-Site Zugriff auf andere Webseite erlauben.
- APIs für Kamera, Mikro, Beschleunigungs- und Magnetfeldsensoren, GPS, Location.

Probleme der Sandbox

Problem 1: Sandbox überwacht Datenpfade überwacht, nicht geflossene Daten.

- Sinnvoller wäre: Darf Steuerdaten übertragen, Kreditkartennummer nicht.
- Selbst dann: Steganographische Codierung.

Problem 2: Fehlende effektive Kontrollen.

- Die meisten Durchbrechungen erlauben offene Einstellungen.
- Diese behindert Implementierungen nicht und findet sich daher (zu) oft.

Problem 3: Pseudokontrollen.

- https, aber die CA hat übersehen, daß der private key gestohlen wurde.

Problem 4: Der Benutzer ist letztverantwortlich, weiß als DAU nicht Bescheid.

Problem 5: Benutzer Override

- Wenn ich https Warnung CA_frm-e7_1818_Suspect_Spoof nicht wegklicke, kann ich ja das Gratispiel nicht downloaden.

Signierter Code

Code kann signiert werden.

Signierter Code erhält weitergehende Rechte in Abhängigkeit vom signierenden Autor.

Probleme:

- Funktioniert nur, wenn Vertrauen in den Code Ersteller gerechtfertigt ist.
- Erfordert entsprechend stringente Identitätsprüfung der CA.

Policies

User kann feingranular steuern, welche Anwendungen welche Rechte erhalten.
Freigabe nach individueller Einschätzung und Anforderung möglich.

Probleme:

- Ist nicht zero install.
- User ist mit kompetenter Entscheidung oft überfordert.
- User kann es nicht richtig beantworten (fehlende Background Info)
- User will es nicht richtig beantworten (verstehet Risiken nicht)
- User muss es immer wieder beantworten (lästig)
- System merkt sich Entscheidung, dem User ist das nicht immer bewußt.

Same Origin Sandbox

Javascript, das in eine Seite der URL x eingebettet ist, kann nur dann

- via XHR auf die URL y zugreifen
- via DOM API auf Seiten der URL y zugreifen

wenn x und y "same origin" sind.

Verschiedener Origin ist:

- Protokoll ist verschieden (http \neq https)
- Port ist verschieden
- Unterschiedliche Subdomänen
- Redirections
- Ein Script von Rechnername, ein anderes von IP Adresse.
Dh: Browser wertet DNS nicht aus

Same Origin Sandbox Spezialfälle

ISP-Anomalität

- `www.myisp.com/dave/*` und `www.myisp.com/john/*` sind same origin.
- Keine Sandbox zwischen verschiedenen Usern bei ISP-Lösungen auf 1 Maschine.

document.domain Property

- **Problem:** `www.bsp.com` und `wiki.bsp.com` sind nicht same origin.
- **Lösung:** Beide Skripts setzen `document.domain = "bsp.com"`.
- **Aber:** Nur eine Verkürzung
der bereits vorhandenen eigenen Domäne möglich.
- **Achtung:** Deprecated.

Erforderlich bei Serviceworker, localStorage, sessionStorage, websockets, IndexedDB, WebSql, Camera, Video, Audio, GPS, Cellid, Filesystem, Extensions, acceleration, magnetic field, Luftdruck, WebRTC, WebData, WebPayment API, Web Authentication API, USB-Schnittstelle, BLE-Schnittstelle, NFC-Schnittstelle, ApplePay.

Weitere Same Origin Regelungen

- **XMLHttpRequest:** SO Restriktion mit Ausnahmemechanismus (**CORS**)
- **local/sessionStorage:** SO Restriktionen
- **indexedDB:** SO Restriktionen
- **Websockets:** Eigener Mechanismus erlaubt Feinabstimmung.
- **Web Messaging:** Eigener Mechanismus erlaubt Feinabstimmung
- **Bilder:** Cross-Site erlaubt.
Kleines Risiko, aber Achtung auf Web Spoofing.
- **Stylesheets:** Cross-Site erlaubt.
Mittleres Risiko durch GUI Redressing.
- **Skripte:** Cross-Site erlaubt.
Hohes Risiko (8ung: Seite übernehmen)
- **Cookies:** Browser-seitig einstellbare Regelungen & Erweiterungen

Dokumente in verschiedenen Fenstern/Tabs können kontrolliert Daten austauschen.

```
1  /* Sender */  
2  targetWindow.postMessage (message, targetURL);  
3  
4  /* Empfänger */  
5  window.addEventListener("message", (event) => {  
6    if (event.origin !== "http://example.org:8080") {return;}  
7    // bearbeite Nachricht in event.data  
8    // antworte ggf. an Fenster event.source  
9  });
```

Src. 1: Web Messaging

Web Messaging (2)

`targetWindow`: Referenz auf das Zielfenster.

- Erhalten beim Öffnen des Zielfensters als return-Wert von `window.open()`
- Erhalten als `contentWindow` Property eines iframes.

`targetURL`: URL des Zielfensters.

- Angabe, um Empfänger weiter einzuschränken; kann "*" sein.

Sendendes Fenster:

- **Gefahr:** Sende vertrauliche Daten an böses Ziel.
- **Abwehr:** Setze `targetURL` auf ein Ziel, dem ich vertraue.

Empfangendes Fenster:

- **Gefahr:** Nutze nicht vertrauenswürdige Daten aus böser Quelle.
- **Abwehr 1:** Nur dann Handler setzen, wenn wirklich empfangen will.
- **Abwehr 2:** Prüfen des `.origin` im event, ob von dieser URL empfangen will.
- **Abwehr 3:** Validiere die erhaltenen Daten

3. Injection

Immer noch einer der häufigsten Angriffe.

1. Übersicht
2. Mobile Code
- 3. Injection**
4. Cross Site
5. Trusted IO Problem

Web 2.0 Techniken zur Generierung dynamischer Seiten erhalten Daten vom Client.

Diese werden dann eingebaut in:

- SQL Queries
- HTML Seiten
- PHP Programme
- Javascript Programme
- Templates

Problem:

- Programmierer macht implizite Annahmen über Semantik und Syntax.
- Wenn diese verletzt werden entstehen unerwünschte Effekte.
- **Angriff 1:** Unerwartete Eingabe in ein Formular durch User.
- **Angriff 2:** Erzeugen von Anfragen (URL, Query, header, body) durch spezifische Angriffsprogramme.

3. Injection

Beispiele für HTML- und SQL-Injection

```
1 $username='Hal<br>lo <script>alert("HUHU!");</script></table></body></html>';
2 $account = 42;
3
4 <html><body><table>
5 <tr><td>User:</td><td>$username</td></tr>
6 <tr><td>Account:</td><td>$account</td></tr>
7 </table></body></html>
```

Src. 2: Was passiert, wenn die entsprechenden Ersetzungen vorgenommen werden?

```
1 $query = SELECT * FROM USERS WHERE user = '$uname';
2 Username:  '; DROP TABLE USERS - -
3
4 Resultat: SELECT * FROM USERS WHERE user=''; DROP TABLE USERS - - '
```

Src. 3: Welche Effekte löst diese Query auf dem DB-Server aus?

3. Injection

HTML-Injection nach xkcd

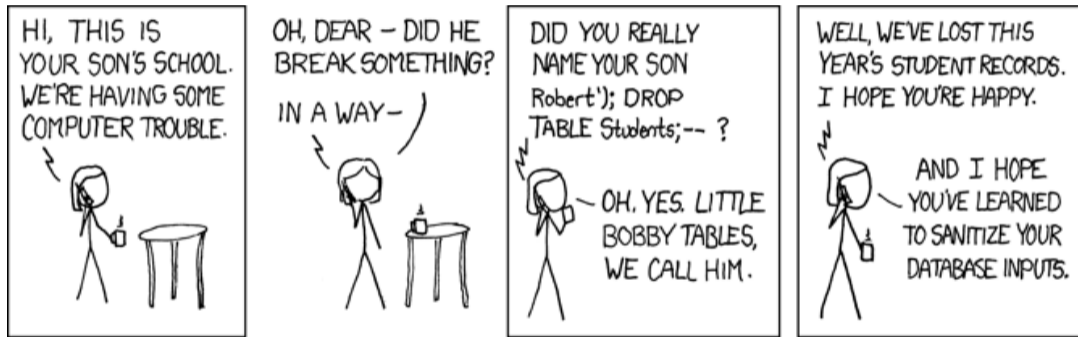


Abb. 3: xkcd327 zu © Rechte siehe Anhang.

Vermeidung von HTML-Injection

```
1 var ele = document.getElementById ("ele");
2 ele.innerHTML = "das ist <b>fett</b>";           // zeigt fett an
3 ele.innerHTML = "das ist &lt;b&gt;fett&lt;/b&gt;"; // zeigt HTML Text an
```

Src. 4: Nutzung von HTML Entities

```
1 function addTextNode(text) {
2     var newtext = document.createTextNode(text),
3     p1 = document.getElementById("p1");
4     p1.appendChild(newtext);
5 }
```

Src. 5: Nutzung von DOM-Funktionen vermeidet HTML-Injection.

Vermeiden von SQL-Injection

- Escapen von Quotes (DB-abhängig).
- Nutzung von Query-Buildern.
- Vermeiden von String Interpolation.
- Typ-Prüfung (String, Name, PLZ)
- Eingabe-Validierung.
- Prepared Statements verwenden.
- PHP Data Objects (PDO) nutzen.
- Separate DB-user für Wartung und Produktion verwenden.
- SQL-seitig reduzierte Rechte für den DB-user vorsehen.

```
1 $stmt = $dbh->prepare ("INSERT INTO user (first, last) VALUES (:fn, :ln)");
2 $stmt -> bindParam(':fn', 'John'); // Korrekte Behandlung von Quotes im Wert
3 $stmt -> bindParam(':ln', 'Doe'); // ist in bindParam gesichert.
4 $stmt -> execute();
```

Src. 6: Prepared Statements.

4. Cross Site

Auch einer der häufigsten Angriffe

1. Übersicht
2. Mobile Code
3. Injection
4. Cross Site
5. Trusted IO Problem

Cross-Site-Request-Forgery

Auslösen einer Web-Transaktion durch Unterschieben einer URL an einen angemeldeten Benutzer.

Beispielhafte Vorgehensweise:

- Client führt Logon auf `bank.example` durch und erhält Session-Cookie.
- Client geht auf andere Webseite.
- Dort: ``
- Browser lädt vermeintliches Bild.
- Browser sendet Session-Cookie mit und löst damit Transaktion aus.
- Bestätigung der Transaktion ist nicht im Bild-Format
- Führt zu *broken image* Ikone.
- Diese wird nicht angezeigt (unsichtbar geschaltet oder off-screen).

Analyse der Problematik

Problem 1: Cookies zur Vermittlung von Rechten.

- Cookies werden bei Request automatisch an die entsprechende Domäne versandt.
- Es gibt keine separaten Checks über den Cookie Versand an den Server.
- Doppelrolle von Cookies: Authentisierung, Tracking, SessionId

Problem 2: Link-Text Semantik

- Text des Links impliziert nicht Funktion des Links.
- Bsp: Link der Geld überweisen als Text anzeigt kann zu ganz anderer URL gehen.

Problem 3: URL Semantik

- Name des Links impliziert nicht Funktion des Links.
- Bsp: URL `https://bsp.de/geldUeberweisen` kann auch das Konto schließen.

Problem 4: Link Aktivierung bedeutet nicht immer User-Interaktion.

- Abruf eines Links auch ohne expliziten Aufruf durch User möglich.
- Bsp: Javascript und `` können Link abrufen.

Reduktion der Angriffsfläche

Reduktion der Angriffsfläche, aber **keine Verhinderung** des Angriffs:

- **Logoff** Feature anbieten & Nutzer zur Verwendung motivieren.
- Session-Cookies nur mit **kurzer Laufzeit** versehen.
Aber: Verkürzt nur Zeitfenster für Angriff.
- Im Server den **referer** header prüfen.
Aber: Kann gefälscht werden.
- Service nur via **POST** anbieten.
Aber: JS kann auch POST absetzen.
- Transaktion **mehrstufig** gestalten.
Aber: Auch mehrstufige CSRF sind möglich.

Verhinderung von CSRF (1)

Unterstützung im Framework:

- Nutzen eines Frameworks mit eingebauter CSRF-Protection.
- **Vorteil:** Funktioniert ohne Nachdenken (meist).
- **Nachteil:** Man versteht keine Details und ist den Framework Fehlern ausgeliefert.

Re-Authentifizierung:

- Benutzer muß sich bei Transaktion erneut authentifizieren.
- **Beispiel:** Bank-Transaktion erfordert Transaktions-basierte TAN.
- **Vorteil:** Sehr sicher.
- **Nachteil:** Zusätzlicher Aufwand für Benutzer.

Verhinderung von CSRF (2)

Synchronizer: Status-behaftete Technik.

- Server sendet Nonce als Token an Client als Teil der Webseite und merkt sich das.
- Client sendet Token wieder zurück als Teil des Requests.
- Server prüft Token.
- Sendung des Tokens wird nur durch Request ausgelöst, der von Webseite stammt.
- Keine automatische Mitsendung des Tokens wie bei jedem Server Request.

Cookie Double Submission: Statusfreie Technik.

- Server sendet der Webseite eine weitere (neue) Nonce als Cookie.
- Request enthält Nonce im Cookie.
- Request enthält Nonce auch als Parameter.
- Server prüft auf Gleichheit.
- Angreifer kann diesen Cookie-Wert nicht lesen und daher auch nicht identisch als Parameter in den Request einbauen.

Content Security Policy (1)

Server setzt Content-Security-Policy Header oder `<meta>`.
Direktiven bestimmen legale Quellen für bestimmte Ressourcen.
Browser befolgt detailliert spezifiziertes Cross-Site Verhalten.

```
1 // Alles nur lokal zu laden
2 default-src 'none'; script-src 'self'; connect-src 'self';
3 img-src 'self'; style-src 'self';
4
5 // Nur gewisse remote Quellen erlaubt:
6 script-src 'self' www.google-analytics.com ajax.googleapis.com;
```

Src. 7: Beispiele für Content Security Policies

Content Security Policy (2)

Ressource Typen Beispiele:

script-src: Woher darf Browser scripts laden
style-src: Woher darf Browser style sheets laden
img-src: Woher darf Browser images laden
connect-src: Woher darf Browser XHR und Websocket Daten holen
font-src: Woher darf Browser Fonts Files laden
media-src: Woher darf Browser video und audio Sourcen laden
frame-src: Woher darf Browser iframes laden

Policy Werte Beispiele:

*	Alle Quellen erlaubt	none	Keine Quellen erlaubt
self	Same Origin	data:	Data URLs
domain.com	Domäne	*.domain.com	Domäne mit Wildcarding
https://cdn.com	Domäne mit https	unsafe-inline	Teil der Seite (style, script tag)

Sehr feingranulare Anpassung für sicherheitsrelevante Sites möglich & wichtig

5. Trusted IO Problem

Ein Kernproblem der Informatik-Sicherheit.

1. Übersicht
2. Mobile Code
3. Injection
4. Cross Site
5. Trusted IO Problem

5. Trusted IO Problem

Beispiel

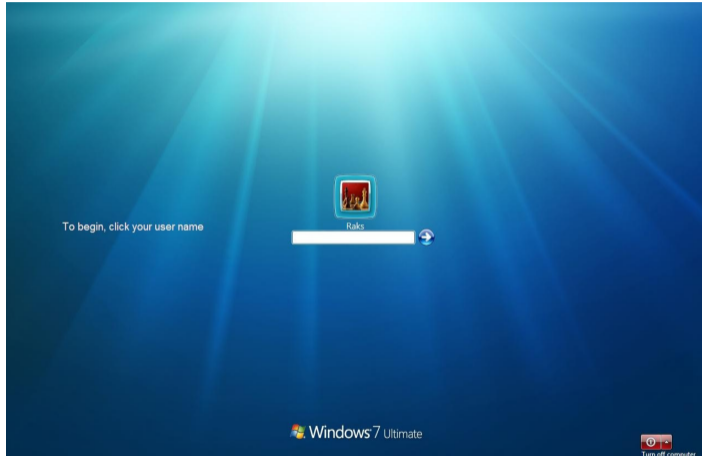


Abb. 4: Ist das ein authentischer Logon-Bildschirm oder ein Bild einer Webseite in Fullscreen Modus?

Web-Cursor Hotspot

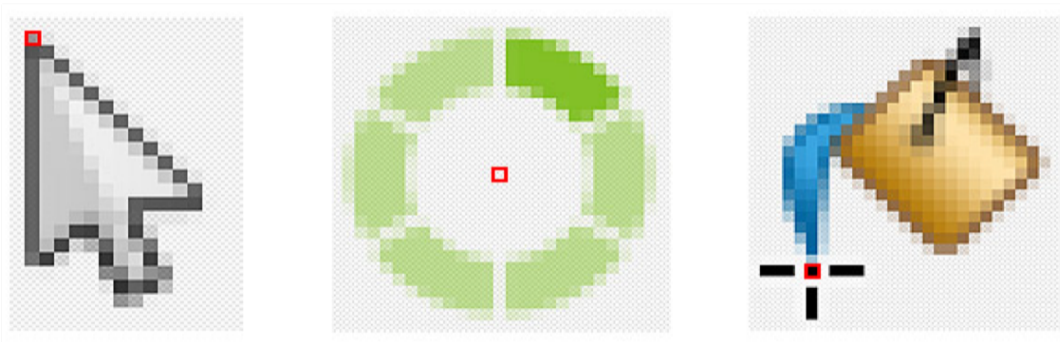


Abb. 5: Ein Web-Cursor hat einen Hotspot für den Mausklick; dieser kann über CSS definiert werden.

GUI Redressing Attacke

Annahme: Eine Webseite sei so gebaut, daß ein Angreifer den Cursor auswechseln kann.

Beispiel für die Annahme:

- Webseite bezieht Inhalte von `www.nachrichten.beispiel.com`
- Webseite bezieht Grafik, Bilder usw. von `www.webgrafik.com`
- Angreifer hat `www.webgrafik.com` geknackt – oder `www.webgrafik.com` ist nur für diesen Angriff prepariert.

Demo:

- Siehe: <https://iuk.one/cursorjacking.html>
- Demo illustriert das Problem mit einem roten Balken.
- In echtem Angriff würde der rote Balken fehlen.
- Aufgabe: Klicken Sie auf "Two" (in die Wortmitte) und denken Sie sich den roten Balken weg!

Web Spoofing (1)

Benutzer ist auf anderer Webseite als er glaubt

Image-Attacke:

- User sieht Bild eines Browsers im Browser statt Browser.
- Damit Location-Zeile beliebig manipulierbar.
- **Lösung:** Location-Zeile immer darstellen.

Web Spoofing (2)

Fullscreen-Attacke:

- In Fullscreen Modus zeigt Browser keine Location-Zeile.
- Damit ist Image-Attacke wieder in Fullscreen möglich.
- **Lösung:** Fullscreen leicht erkennbar machen.

Homographie-Attacke:

- Cyrillisches "а" (U+0430) und ASCII "a" (U+0061) sind unterschiedliche Zeichen
- sehen aber in der Font der Location-Bar gleich aus.
- **Lösung:** UTF in Punycode codieren.

Punycode-Attacke:

- Punycode ist eine ASCII-basierte Darstellung von Unicode.
- **Beispiel:** München ist Mnchen-3ya.
- **Beispiel:** [xn--80ak6aa92e.com](#) ist Punycode für apple.com (Link ausprobieren!)
- Wenn Browser Punycode als UTF-Original darstellt, dann sind homographische Angriffe möglich.

5. Trusted IO Problem

Mixed Content

Eine Mixed Content Seite ist eine Seite die http und https einbindet.

Passive mixed content: audio, img, video, object.

Active mixed content: script, link (css), XHR, iframe, object.

Problem:

- Seite lädt unter https und Benutzer fühlt sich abgesichert.
- Seite lädt Bilder, CSS, Scripte, iFrames, XHR, Websockets usw. über http.
- Seite wird dann darüber manipuliert.

Beachte: Browser gehen zunehmend dazu über, mixed content zu blockieren.

Subresource Integrity: Problem

Problem:

- Sind die Teilressourcen einer Webseite was sie aus Betreibersicht sein sollen?
- **Speziell:** Was liefern CDNs genau aus?
- **Idee:** Gebe dem CDN meinen private key.
- **Aber:** Jetzt kann CDN meine Seite manipulieren.

```
1 <script src="https://example.com/example-framework.js"  
2 integrity="sha384oqVuAfXRRkap7fdgcCY5uykM6+R9GqQ8K/uxy9rx7HNQlGY11kPzQho1wx4JwY8wC"  
3 crossorigin="anonymous"></script>
```

Src. 8: Die einbettende Seite sagt im einbettenden Tag, welchen hash Wert die zu ladende Ressource haben soll und von welchen cross origins sie geladen werden darf.

Subresource Integrity: Weiterer Lösungsansatz

Restproblem:

- Ist eine resource mit diesem Hash-Wert sicher?

Lösungsansatz:

- Gitlab / npm & open source registry hat Code.
- CVE (Common Vulnerabilities and Exposures) Datenbank hat Fehlerberichte.
- Weitere DB haben positive Reviews.
- Blockchain bindet diese Informationen zusammen.
- Browser Extension cached Whitelist und Blacklist von subresource integrity hashes.

Publiziert in: C. H. Cap, B. Leiding: Ensuring Resource Trust and Integrity in Web Browsers Using Blockchain Technology

HTTP Strict Transport Security

Server Header, der Browser anweist, für diese Site künftig https zu nutzen.

Erlaubt Attribute zur

- Steuerung der zeitlichen Wirksamkeit dieser Anweisung.
- Ausweitung dieser Anweisung auf Subdomänen.

Anwendung:

- Verhindert https nach http downgrade Attacke.
- Verhindert Weiterleitung von https auf http.
- Header ignoriert, wenn Seite via http geladen.
Konsequent (Angreifer kann http Seite ohnehin manipulieren)

Info: [MDN über Strict Transport Security](#)

Anhang

Übersicht

Programmquellenverzeichnis

Prog

Verzeichnis aller Abbildungen

Abb

Rechtsnachweise

©

Rechtliche Hinweise

§

Zitierweise dieses Dokuments

→

Verzeichnis aller Folien

📖

1	Web Messaging	17
2	HTML Injection	21
3	SQL Injection	21
4	Nutzung von HTML Entities	23
5	DOM-Funktionen	23
6	Prepared Statements	24
7	Beispiele für Content Security Policies	31
8	Subresource Integrity Lösung	40

1	4
2	5
3	xkcd327 zu	22
4	Logon Screen oder Fullscreen?	34
5	Cursor Hotspot	35

Abb. 1 Quelle: Wordfence

Abb. 2 Quelle: <https://www.ptsecurity.com/ww-en/analytics/web-vulnerabilities-2020/>

Abb. 3 Source: <https://xkcd.com/327/> lizenziert nach <http://creativecommons.org/licenses/by-nc/2.5/>

Rechtliche Hinweise (1)

Die hier angebotenen Inhalte unterliegen deutschem Urheberrecht. Inhalte Dritter werden unter Nennung der Rechtsgrundlage ihrer Nutzung und der geltenden Lizenzbestimmungen hier angeführt. Auf das Literaturverzeichnis wird verwiesen. Das **Zitatrecht** in dem für wissenschaftliche Werke üblichen Ausmaß wird beansprucht. Wenn Sie eine Urheberrechtsverletzung erkennen, so bitten wir um Hinweis an den auf der Titelseite genannten Autor und werden entsprechende Inhalte sofort entfernen oder fehlende Rechtsnennungen nachholen. Bei Produkt- und Firmennamen können Markenrechte Dritter bestehen. Verweise und Verlinkungen wurden zum Zeitpunkt des Setzens der Verweise überprüft; sie dienen der Information des Lesers. Der Autor macht sich die Inhalte, auch in der Form, wie sie zum Zeitpunkt des Setzens des Verweises vorlagen, nicht zu eigen und kann diese nicht laufend auf Veränderungen überprüfen.

Alle sonstigen, hier nicht angeführten Inhalte unterliegen dem Copyright des Autors, Prof. Dr. Clemens Cap, ©2020. Wenn Sie diese Inhalte nützlich finden, können Sie darauf verlinken oder sie zitieren. Jede weitere Verbreitung, Speicherung, Vervielfältigung oder sonstige Verwertung außerhalb der Grenzen des Urheberrechts bedarf der schriftlichen Zustimmung des Rechteinhabers. Dieses dient der Sicherung der Aktualität der Inhalte und soll dem Autor auch die Einhaltung urheberrechtlicher Einschränkungen wie beispielsweise **Par 60a UrhG** ermöglichen.

Die Bereitstellung der Inhalte erfolgt hier zur persönlichen Information des Lesers. Eine Haftung für mittelbare oder unmittelbare Schäden wird im maximal rechtlich zulässigen Ausmaß ausgeschlossen, mit Ausnahme von Vorsatz und grober Fahrlässigkeit. Eine Garantie für den Fortbestand dieses Informationsangebots wird nicht gegeben.

Die Anfertigung einer persönlichen Sicherungskopie für die private, nicht gewerbliche und nicht öffentliche Nutzung ist zulässig, sofern sie nicht von einer offensichtlich rechtswidrig hergestellten oder zugänglich gemachten Vorlage stammt.

Use of Logos and Trademark Symbols: The logos and trademark symbols used here are the property of their respective owners. The YouTube logo is used according to brand request 2-9753000030769 granted on November 30, 2020. The GitHub logo is property of GitHub Inc. and is used in accordance to the GitHub logo usage conditions <https://github.com/logos> to link to a GitHub account. The Tweedback logo is property of Tweedback GmbH and here is used in accordance to a cooperation contract.

Disclaimer: Die sich immer wieder ändernde Rechtslage für digitale Urheberrechte erzeugt für mich ein nicht unerhebliches Risiko bei der Einbindung von Materialien, deren Status ich nicht oder nur mit unverhältnismäßig hohem Aufwand abklären kann. Ebenso kann ich den Rechteinhabern nicht auf sinnvolle oder einfache Weise ein Honorar zukommen lassen, obwohl ich – und in letzter Konsequenz Sie als Leser – ihre Leistungen nutzen.

Daher binde ich gelegentlich Inhalte nur als Link und nicht durch Framing ein. Lt EuGH Urteil 13.02.2014, C-466/12 ist das unbedenklich, da die benutzten Links ohne Umgehung technischer Sperrern auf im Internet frei verfügbare Inhalte verweisen.

Wenn Sie diese Rechtslage stört, dann setzen Sie sich für eine Modernisierung des völlig veralteten Vergütungssystems für urheberrechtliche Leistungen ein. Bis dahin klicken Sie bitte auf die angegebenen Links und denken Sie darüber nach, warum wir keine für das digitale Zeitalter sinnvoll angepaßte Vergütungssysteme digital erbrachter Leistungen haben.

Zu Risiken und Nebenwirkungen fragen Sie Ihren Rechtsanwalt oder Gesetzgeber.

Weitere Hinweise finden Sie im Netz [hier](#) und [hier](#) oder [hier](#).

Zitierweise dieses Dokuments

Wenn Sie Inhalte aus diesem Werk nutzen oder darauf verweisen wollen, zitieren Sie es bitte wie folgt:

Clemens H. Cap: Web 2.0 und Sicherheit. Electronic document. <https://iuk.one/1066-1011> 5. 7. 2021.

Bibtex Information: <https://iuk.one/1066-1011.bib>

```
@misc{doc:1066-1011,  
  author      = {Clemens H. Cap},  
  title       = {Web 2.0 und Sicherheit},  
  year        = {2021},  
  month       = {7},  
  howpublished = {Electronic document},  
  url         = {https://iuk.one/1066-1011}  
}
```

Typographic Information:

Typeset on July 5, 2021

This is pdfTeX, Version 3.14159265-2.6-1.40.21 (TeX Live 2020) kpathsea version 6.3.2

This is pgf in version 3.1.5b

This is preamble-slides.tex myFormat©C.H.Cap

Titelseite	1
1. Übersicht	
Übersicht	3
Klassifikation von Sicherheitsproblemen	4
Klassifikation von Sicherheitsproblemen	5
Ziele des Angreifers	6
2. Mobile Code	
Sicherheitsproblematik bei Mobilem Code	8
Sandbox (1)	9
Sandbox (2)	10
Probleme der Sandbox	11
Signierter Code	12
Policies	13
Same Origin Sandbox	14
Same Origin Sandbox Spezialfälle	15
Weitere Same Origin Regelungen	16
Web Messaging (1)	17
Web Messaging (2)	18
3. Injection	
Problematik	20
Beispiele für HTML- und SQL-Injection	21
HTML-Injection nach xkcd	22
Vermeidung von HTML-Injection	23
Vermeiden von SQL-Injection	24




4. Cross Site

Cross-Site-Request-Forgery	26
Analyse der Problematik	27
Reduktion der Angriffsfläche	28
Verhinderung von CSRF (1)	29
Verhinderung von CSRF (2)	30
Content Security Policy (1)	31
Content Security Policy (2)	32

5. Trusted IO Problem

Beispiel	34
Web-Cursor Hotspot	35
GUI Redressing Attacke	36
Web Spoofing (1)	37
Web Spoofing (2)	38
Mixed Content	39
Subresource Integrity: Problem	40
Subresource Integrity: Weiterer Lösungsansatz	41
HTTP Strict Transport Security	42

Legende:

-  Fortsetzungsseite
-  Seite ohne Überschrift
-  Bildseite